Homework 5:
Classification using Decision Tree and Random Forest

Submit your assignments on Gradescope.
Please name your coding assignment as 'HW5.py'.
Use the provided Python template file, and complete the functions ONLY. (DO NOT edit function definitions, code outside the function, or use other libraries).

In this homework, you have been provided with the Car evaluation dataset to implement a decision tree classifier from scratch. The functions that you will be graded on have pre-initialized variables as a guide for the data types allowed in the return statements. You will finally evaluate the performance and visualize the decision tree.

The project starts with loading the Car Evaluation dataset from a local file and converting categorical variables to numerical. The data is then split into training and testing set. The decision tree model is fit on the training set for different hyperparameters and the model with best accuracy is finally selected to make predictions on the test dataset. The code for visualizing the results are provided in the starter template.
A code to visualize the tree structure of the decision tree is also provided for better understanding.

Implementing Decision Tree from Scratch

In this part, you will implement a decision tree classifier from scratch. The main functions you need to implement include the following functions. More details regarding data types of parameters are provided in the starter code. You need to complete :

- **entropy(x):** Compute the entropy of a dataset.
- **TreeRegressor** class: Implements the decision tree.

Implementing Random Forest

In this part, you will implement a random forest classifier using the decision tree classifier you have built. A random forest is an ensemble method that fits multiple decision trees on different subsets of the dataset and averages their predictions. The **RandomForestRegressor** class implements the random forest.

The parameters for this class are:

- **n_estimators**: The number of trees in the forest.
- **max_depth**: The maximum depth of the trees.
- **min_samples_split**: The minimum number of samples required to split a node.
- **n_features**: The number of features to consider when looking for the best split.

The main methods for this class include:

- **fit method**: Fits multiple decision trees on different bootstrap samples of the dataset.
- **predict method**: Predicts the labels for a given dataset X by averaging the predictions of the individual trees.

Tasks

1. **Implement the entropy function to measure the impurity of a split.**

2. ***Tree Regressor***
   It includes methods for fitting the model to training data, predicting labels for new data, and building the tree structure based on the best feature splits.
   The parameters for this class are
   - **min_samples_split**: The minimum number of samples required to split a node.
   - **max_depth**: The maximum depth of the tree.
   - **n_features**: The number of features to consider when looking for the best split.
   - **root**: The root node of the decision tree, initially set to None

   fit method:
   Builds the decision tree by calling the build_tree method and sets the root of the tree.

predict method:
Predicts the labels for a given dataset X by traversing the tree starting from the root for each data point.

Implement the following methods to complete the class:

build_tree method:
Recursively builds the decision tree. It checks the stopping criteria (maximum depth, minimum samples, or only one label) and creates leaf nodes accordingly. Otherwise, it finds the best feature and threshold to split the data and recursively builds the left and right subtrees.

get_best_ split method:
Finds the best feature and threshold to split the data by calculating the information gain for each feature and threshold. It returns the feature index and threshold that result in the highest information gain.

information_gain method
Calculates the information gain from splitting the data at a given threshold for a feature. Information gain is the difference in entropy before and after the split.

split method:
Splits the data based on a given threshold. It returns the indices of the samples that go to the left and right nodes.

traverse_tree method:
Traverses the tree from the root to a leaf node based on the feature values of a data point x. It returns the value of the leaf node.

common_thing method:
Finds the most common label in a list of labels. This is used to assign a value to leaf nodes when the stopping criteria are met.

3. **RandomForestRegressor class**

fit method
This method creates a list of individual decision tree models. For each tree, generate a bootstrap sample of the training data. This means randomly sampling with replacement from the training data. Train a decision tree on each bootstrap sample. You can use the TreeRegressor class for this. Append each trained tree to the list of trees in the forest.

predict method
For each instance in the input data, collect predictions from all the trees in the forest. Then combine the predictions from all the trees and use the majority vote (mode) of the predictions for classification.

This is a report assignment.

- Plot the graph of accuracy values of the decision tree against the tree depth. Discuss your resulting graphs. What is the highest accuracy you can obtain and for what depth.

- Plot the graph of accuracy values against the number of estimators for the random forest. Discuss your resulting graphs. What is the highest accuracy you can obtain and for how many estimators.

- Compare the performance of the decision tree and the random forest. Discuss which model performs better and why.

- Include visualizations that help in understanding the performance differences between the decision tree and the random forest.

- Discuss the results obtained from both the decision tree and random forest models. Provide insights into why certain models perform better and how hyperparameters affect their performance.